ঙ

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/620,534	07/16/2003	Hoi Chang	P00620-US-00 (19232.0003)	8981

22446 7590 03/12/2007

ICE MILLER LLP
ONE AMERICAN SQUARE, SUITE 3100
INDIANAPOLIS, IN 46282-0200

EXAMINER	
KIM, JUNG W	

ART UNIT	PAPER NUMBER
2132	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	03/12/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No. 10/620,534	Applicant(s) CHANG ET AL.	
	Examiner Jung Kim	Art Unit 2132	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *02 January 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *67-72,78-88,90-92,94,95,101-108,111-114 and 116-119* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☒ Claim(s) *94,95,104 and 105* is/are allowed.

6)☒ Claim(s) *67-72,78-85,88,90-92,101-103,106-108,111-114 and 116-119* is/are rejected.

7)☒ Claim(s) *86 and 87* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on 1/02/07.

2. Claims 67-72, 78-88, 90-92, 94, 95, 101-108, 111-114 and 116-119 are pending.

3. Claim 119 is new.

Response to Arguments

4. Applicant's arguments, with respect to the 112/1st paragraph rejections have been fully considered and are persuasive. The 112/1st paragraph rejections of claims 67-72 and 78-87 have been withdrawn.

5. Regarding applicant's arguments that the limitations of claims 78, 80 and 81 are not disclosed by Johnson, examiner disagrees. In particular, applicant alleges that "Johnson does not teach selecting and revising a computation, but rather teaches generating multiple copies of a computation in "checking cascades" and adding a new statement (an IF statement) into the program to compare the computed outputs ... Johnson does not disclose or teach use of expected values, but rather the insertion of multiple computations producing intermediate results, and intertwining the added computations and intermediate results." Remarks, p. 35, first paragraph, last sentence; p. 38, first sentence. However, the portion of Johnson identified in the rejections explicitly define such steps. On col. 10, line 14- col. 11, line 18, Johnson discloses taking a computation (a + b) and changing the domain of the computation by applying an intertwining function "to increase the density of interdependencies." Col. 10:3.

Output of the resulting EIS encoding is therefore dependent on the runtime value of the initial input values a and b. Johnson further discloses inserting checks and traps on the various computed values, and if the expected value is different from the runtime value, causing the software program to execute incorrectly. Col. 12:10-14. Hence, contrary to applicant's allegations, Johnson discloses all the features of claims 78, 80 and 81.

6. With respect to applicant's arguments that Johnson does not disclose the limitations of claim 82, in particular that "Johnson does not teach or disclose 'selecting a program block that computes a results necessary for proper execution,' 'determining the expected value of the silent guard' and 'installing a first computation dependent on the silent guard ... such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the first computation causes the result computed by the program block to evaluate improperly' ... making a 'result necessary for proper execution' and causing 'the results computed by the program block to evaluate improperly' using a silent guard and a computation" (Remarks, pg. 39, last paragraph), examiner disagrees. On the contrary, Johnson discloses selecting a computation of the software program (col. 10:16; 12:11-14), which anticipates the limitation "selecting a program block that computes a results necessary for proper execution," then inserting a check and trap, such that the check identifies the expected value of a cascades computed value (col. 12:12), which anticipates the limitation "selecting a silent guard for the program block; determining the expected value of the silent guard at the start of execution of the program block," and if the cascades computed values are different from expected, set the TamperFlag and start a time, and if the TamperFlag is set and the

timer has expired, go to trap code (col. 12:13), which anticipates the limitation "installing a branch instruction dependent on the silent guard in the software program, such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the branch instruction causes the program block to be skipped, causing the software program to execute improperly." Hence, Johnson anticipates all limitations of claim 82.

7. Applicant's arguments, with respect to the 102(b) rejections of claims 86 and 87 have been fully considered and are persuasive. The 102(b) rejections of claims 86 and 87 have been withdrawn.

8. Applicant's arguments that Johnson does not disclose the limitations of claims 91 and 106 are not persuasive; applicant's duty to view of the whole of the teachings of Johnson explicitly ignores relevant portions of the disclosure as it pertains to these claims. (The only literal differences in claims 91 and 106 are that claim 91 recites an invention with respect to a first dependency point, whereas claim 106 recites the invention with respect to a first and second dependency point. However, claim 106 never distinguishes the first and second points as separate points; hence, claim 106 discloses all limitations of claim 91) Johnson anticipates the limitations as follows: Johnson discloses inserting a plurality of cascades, and inserting a plurality of checks as needed, such that if the cascades computed values are different than expected, set the TamperFlag and start a timer (col. 11:55-col. 12:14), which anticipates the limitation "a silent guard variable having an expected value at a first dependency point in the software program; a mathematical computation that includes the runtime value of the

silent guard variable and an expected term, the expected term being set based on the expected term of the silent guard variable at the first dependency point;" moreover, because the checks are inserted after a given cascade, and the fact that a plurality of cascades are inserted as well as a plurality of checks, any cascade following a check is dependent on result of the previous check, which anticipates the limitation "wherein the runtime value of the program variable at the second dependency point is dependent on the result of the mathematical computation which is dependent on the runtime value of the silent guard variable at the first dependency point, such that the runtime value of the program variable at the second dependency point will not equal the expected value of the program variable at the dependency point if the runtime value of the silent guard variable at the first dependency point does not equal the expected value of the silent guard variable at the first dependency point, which will cause the software program to execute improperly." Hence, Johnson anticipates all limitations of claims 91 and 106.

9. Regarding applicant's argument that Johnson does not disclose the limitations of amended claim 112, which incorporates the limitations of claims 109 and 110, examiner disagrees. Applicant argues that "Johnson does not teach, disclose or suggest a mathematical computation that makes the TamperFlag dependent on the runtime value of the program variable wherein the result the mathematical computation is corrupted if the runtime value of the program variable is not equal to the expected value of the program variable" (Remarks, pgs. 43-44). However, the setting of the TamperFlag is dependent on the runtime value of the cascades computed values (col. 12:11), which depends on the runtime value of the program values selected for the intertwining

function. (9:35-11:17) If the runtime values of these program values do not match the expected runtime values, the checks on the cascades computed values would result in the TamperFlag being set. Hence, Johnson discloses the aforementioned limitation of claim 112.

10. Applicant's arguments that Johnson does not disclose the limitations of claim 114 as amended are not persuasive. The rejection of claim 114 in the 102(b) rejection section below outlines how Johnson anticipates these new limitations.

11. Applicant's argument that Johnson does not disclose the limitations of amended claim 117, which incorporates the limitations of claims 114 and 115, are not persuasive. Johnson discloses identifying program values and using an intertwining function to create interdependences (9:35-11:17), which anticipates the limitation " identifying a program variable in the software program having an expected value at an insertion point;" inserting a check to determine if a cascades computed values differ from expected, and if the computed values differ from expected, setting a TamperFlag and starting a timer; and when the Tamperflag is set, go to trap code (12:10-14), which anticipates the limitation "adding a program block for causing execution of the software program, the program block including at least one program instruction; inserting a branch instruction at the start of the execution of the program block; making the branch instruction dependent on a silent guard having an expected value; making the runtime value of the silent guard dependent on the runtime value of the program variable, wherein the result of the mathematical computation is corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the

insertion point, such that the runtime value of the silent guard equals the expected value of the silent guard if the runtime value of the program variable equals the expected value of the program variable at the insertion point, and the branch instruction causes the program block to be executed if the runtime value of the silent guard is not equal to the expected value of the silent guard." Hence, Johnson anticipates all limitations of claim 117.

12. With respect to applicant's argument that Collberg does not disclose determining an expected value of a silent guard and/or an expected value of a program variable (Remarks, pg. 53, last sentence; pg. 54, 1st full paragraph; pg. 55, last paragraph), examiner disagrees. For example, Collberg discloses an obfuscation transformation wherein the variable x is assigned 2*a1 + a2 (fig. 18, step 8'). The variable x necessarily has the following expected value: x == 1; if x was not equal to 1 or 2, the obfuscated program would not operate properly, i.e. the "if" statement would not branch according to the logic of the program prior to obfuscation transformation. Hence, Collberg discloses an expected value of a silent guard. Moreover, because the program variables are translated into variables a1, a2, b1, b2, c1 and c2, the program variables necessarily have expected values. If the program variables have a runtime value that was different than the expected value, then the transformed program would not operate as intended. Finally, the translation of the program variables A, B, C to variables a1, b1, b2, c1 and c2, and the corresponding operations on variables a1, a2, b1, b2, c1 and c2 suggest a mathematical expression including the runtime value of the selected program variable and the expected value of the selected program variable. Hence, Collberg also

discloses an expected value of the program variable. Therefore, the rejections of claims 67-69 are deemed proper.

13. In reply to applicant's argument that Collberg does not disclose the limitations of claims 71 and 72, in particular that Collberg does not disclose "a mathematical expression that ... evaluates to the expected value of the computation variable if the runtime value of the silent guard variable is equal to the expected value of the silent guard variable;" (pg. 57, 1st paragraph) "inserting a mathematical expression including the runtime value of the silent guard variable and the expected value of the silent guard variable into the selected computation," (pg. 57, 3rd paragraph); examiner respectfully disagrees. For example, Collberg discloses an obfuscation transformation wherein the variable x is assigned 2*a1 + a2 (fig. 18, step 8'). The variable x necessarily has an expected value: x == 1; if x was not equal to 1 or 2, the obfuscated program would not operate properly, i.e. the "if" statement would not branch according to the logic of the program prior to obfuscation. Hence, Collberg discloses a mathematical expression that ... evaluates to the expected value of the computation variable if the runtime value of the silent guard variable is equal to the expected value of the silent guard variable; and inserting a mathematical expression including the runtime value of the silent guard variable and the expected value of the silent guard variable into the selected computation. Therefore, the rejections of claims 71 and 72 are deemed proper.

14. Applicant's arguments with respect to the rejections of claims 79 and 88 are analogous to the arguments against the rejections of claims 78 and 67-69; hence the

rejections of claims 79 and 88 are deemed proper for the same reasons outlined with respect to the rejections of claims 78 and 67-69.

15. Applicant's arguments with respect to the remaining rejections of the claims are based on the arguments of the rejections to the claims outlined above; hence the rejections of these claims are deemed proper for the same reasons elaborated with respect to the rejections of claims discussed above.

Claim Rejections - 35 USC § 102

16. Claims 78, 80-85, 91, 92, 101-103, 106-108, 111-114 and 116-119 are rejected under 35 USC 102(b) as being anticipated by Johnson USPN 5,748,741 (hereinafter Johnson).

17. As per claim 78, Johnson discloses a computer implemented method for adding tamper resistance to a software program, the method comprising;

a. selecting a program variable in the software program (col. 10:29);

b. selecting a computation in the software program (col. 10:16; 12:11-14: the code following insertion of the check);

c. determining an expected value of the program variable at the point of execution of the selected computation (col. 12:10); and

d. revising the selected computation to be dependent on the runtime value of the program variable, such that the software program executes incorrectly if the

runtime value of the program variable is not equal to the expected value of the program variable (col. 12:11-14).

18. As per claim 80, Johnson further discloses the step of revising the selected computation comprises: selecting a computation variable used in the selected computation; determining an expected value of the computation variable at the point of execution of the selected computation; and replacing the computation variable with a mathematical expression that is dependent on the runtime value of the program variable, such that the mathematical expression evaluates to the expected value of the computation variable if the runtime value of the program variable is equal to the expected value of the program variable (col. 10:42-11:18).

19. As per claim 81, Johnson further discloses the step of revising the selected computation comprises: inserting a mathematical expression including the runtime value of the program variable and the expected value of the program variable into the selected computation (col. 10:42-58).

20. As per claim 82, Johnson discloses a computer implemented method for adding tamper resistance to a software program, the method comprising:

e. selecting a program block that computes a result necessary for proper execution of the software program, the program block comprising at least one

program instruction (col. 10:16; 12:11-14: the code following insertion of the trap);

f. selecting a silent guard for the program block; determining the expected value of the silent guard at the start of execution of the program block (col. 12:12); and

g. installing a first computation dependent on the silent guard in the software program, such that if the runtime value of the silent guard is not equal to the expected value of the silent guard then the first computation causes the result computed by the program block to evaluate improperly, causing the software program to execute improperly (col. 12:13).

21. As per claim 83, Johnson further discloses the step of selecting a silent guard comprises: adding a silent guard variable to the software program; using the silent guard variable as the silent guard; and installing an initialization instruction for the silent guard variable that executes prior to the first computation in the software program, the initialization instruction setting the silent guard variable equal to the expected value of the silent guard (col. 11:18).

22. As per claim 84, Johnson further discloses the step of selecting a silent guard comprises: selecting a program variable in the software program; and using the program value as the silent guard (col. 10:29 and 10:42-11:7).

23. As per claim 85, Johnson further discloses the step of selecting a silent guard comprises:

h. selecting an insertion point in the software program; selecting a program variable in the software program; determining the expected value of the program variable at the insertion point; making the runtime value of the silent guard dependent on the runtime value of the program variable, such that the runtime value of the silent guard equals the expected value of the silent guard if the runtime value of the program variable equals the expected value of the program variable at the insertion point (col. 10:29-11:18).

24. As per claim 101, Johnson further discloses the program variable is used in more than one instruction of the software program, and the last instruction to use the program variable before the point of execution of the selected computation during execution of the software program is separated from the point of execution of the selected computation by a plurality of program instructions not using the program variable. (col. 12:22-24 and lines 25-27)

25. As per claim 102, Johnson further discloses the installation point of the initialization instruction for the silent guard variable is separated from the installation point of the branch instruction by a plurality of program instructions of the software program. (col. 12:22-24 and lines 25-27)

26. As per claim 103, Johnson further discloses the insertion point is separated from the installation point of the branch instruction by a plurality of program instructions of the software program. (col. 12:22-24 and lines 25-27)

27. As per claims 106-108, Johnson discloses a recordable computer media having a tamper resistant software program recorded thereon, the tamper resistant software program comprising:

i. a silent guard variable having an expected value at a first dependency point in the software program; a program variable having an expected value at a second dependency point in the software program; a mathematical computation that includes the run time value of the silent guard variable and an expected term, the expected term being set based on the expected value of the silent guard variable at the first dependency point; wherein the runtime value of the program variable at the second dependency point is dependent on the result of the mathematical computation which is dependent on the runtime value of the silent guard variable at the first dependency point, such that the runtime value of the program variable at the second dependency point will not equal the expected value of the program variable at the second dependency point if the runtime value of the silent guard variable at the first dependency point does not equal the expected value of the silent guard variable at the first dependency point, which will cause the software program to execute improperly; wherein the dependency point is separated from the second dependency point by a plurality of program

instructions; and the tamper resistant software program further comprising an instruction setting the value of the program variable at the second dependency point equal to a function of the runtime value of the program variable and the result of the mathematical computation. Col. 10:3-11:18.

28. As per claim 112, Johnson discloses a recordable computer media having a tamper resistant software program recorded thereon, the tamper resistant software program comprising:

j. a program block for causing improper execution of the software program, the program block including at least one program instruction (col. 12:14);

k. a silent guard in the software program having an expected value at the start of execution of the program block (12:11); and

l. a program variable having an expected value at an insertion point (col. 10:29);

m. wherein the runtime value of the silent guard is dependent on the runtime value of the program variable using a mathematical computation that includes the runtime value of the program variable, wherein the result of the mathematical computation is corrupted if the runtime value of the program variable is not equal to the expected value of the program variable at the insertion point (col. 10:42-49 and line 62; 11:3 and line 18); such that the runtime value of the silent guard equals the expected value of the silent guard if the runtime value of the program

variable equals the expected value of the program variable at the insertion point (col. 10:42-11:18; 12:11); and

n. a branch instruction in the software program dependent on the runtime value of the silent guard, wherein the branch instruction will cause the program block to be executed if the runtime value of the silent guard is not equal to the expected value of the silent guard, causing the program to execute improperly (12:11-14).

29. As per claim 111, Johnson further discloses the insertion point is separated from the branch instruction by a plurality of program instructions. (col. 12:22-24 and lines 25-27)

30. As per claim 113, Johnson further discloses the mathematical computation includes an expected term, the expected term being set based on the expected value of the program variable at the insertion point. (col. 10:42-11:14)

31. As per claim 114, Johnson further discloses a method for adding tamper resistance to a software program, the method comprising:

o. determining a program block in the software program having a program variable with an expected value that causes improper execution of the software program when the program variable is not equal to the expected value of the

program variable, the program block including at least one program instruction; (col. 10:2-11:17; col. 11:55-67; inserting a plurality of cascades)

p. inserting a silent guard having an expected value at the start of execution of the program block; (col. 12:10)

q. making the program variable dependent on the silent guard using a mathematical function, such that the mathematical function does not compute the expected value of the program value if the silent guard is not equal to the expected value of the silent guard, which causes the software program to execute improperly. (checks are inserted after a given cascade, moreover the fact that a plurality of cascades are inserted as well as checks, any cascade following a check is dependent on a previous check, i.e. if the cascade computed values are different from expected then go to trap code)

32. As per claims 116-118, they are method claims corresponding to claims 109-113, and they do not teach or define above the information claimed in claims 109-113. Therefore, claims 116-118 are rejected as being unpatentable as being anticipated by Johnson over for the same reasons set forth in the rejections of claims 109-113.

33. As per claims 91 and 92, they are method corresponding to claims 106-108, and they do not teach or define above the information claimed in claims 106-108. Therefore, claims 91 and 92 are rejected as being unpatentable as being anticipated by Johnson over for the same reasons set forth in the rejections of claims 106-108.

34. As per claim 119, Johnson further discloses wherein the mathematical computation includes the runtime value of the silent guard and the expected value of the silent guard. (col. 12:11-14)

Claim Rejections - 35 USC § 103

35. Claims 67-72, 79, 88 and 90 rejected under 35 U.S.C. 102(a) as being unpatentable over Collberg "A Taxonomy of Obfuscation Transformations" (hereinafter Collberg) in view of Johnson.

36. As per claim 67, Collberg discloses a computer implemented method for adding tamper resistance to a software program (pg. 18, section 7.1.3; pg. 19, fig. 18(a-e)), the method comprising:

r. adding a silent guard variable to the software program (pg. 19, fig. 18(a-c), fig. 18(e), the variable "x");

s. selecting a computation in the software program (pg. 19, fig. 18(e), steps 1-10);

t. determining an expected value of the silent guard variable at the execution point of the selected computation (pg. 19, fig. 18(e), steps 5', 7' and 8': assignment of x;

u. setting the runtime value of the silent guard variable to the expected value of the silent guard variable in the software program at a silent guard insertion point; and

v. revising the selected computation to be dependent on the runtime value of the silent guard variable, such that the selected computation will evaluate improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable ()pg. 19, fig. 18(e), steps 5', 7' and 8': assignment of c1 and c2.

37. In the example of figure 18, Collberg does not expressly disclose the silent guard insertion point being separated from the execution point of the selected computation by a plurality of program instructions of the software program. However, in a different section, Collberg discloses inserting dead or irrelevant code into a basic block, and also interleaving methods to introduce transformations to increase the complexity of the code and hence enhance the level of obfuscation. (pg. 11, section 6.2.1 "Insert Dead or Irrelevant Code"; pg. 15, section 6.3.2, "Interleave Methods") Therefore, it would be obvious to one of ordinary skill in the art at the time the invention was made for the silent guard insertion point to be separated from the execution point of the selected computation by a plurality of program instructions of the software program. One would be motivated to do so to increase the level of obfuscation of the code as taught by Collberg, ibid.

38. Moreover, in the example of figure 18, Collberg discloses that the selected computation will evaluate improperly if the runtime value of the silent guard variable is

not equal to the expected value of the silent guard variable (as noted above), but does not disclose that the software executes improperly. Johnson discloses a method of tamperproofing software by introducing cascades and intertwining blocks. One technique taught by Johnson relevant to the limitation of the instant claim is a feature of checking whether the computed values are different from the expected values, and if not then a TamperFlag is set and a timer is started; when the timer expires, operation of the software program terminates at a trap code to prevent tampering of the software code. (col. 12:13-15) Therefore, it would be obvious to one of ordinary skill at the time the invention was made for the software program to execute improperly if the runtime value of the silent guard variable is not equal to the expected value of the silent guard variable to prevent tampering of the software. The aforementioned cover the limitations of claim 67.

39. As per claim 68, the rejection of claim 67 under 35 USC 103(a) as being unpatentable over Collberg in view of Johnson is incorporated herein. (supra) Collberg further discloses: selecting a program variable in the software program; determining an expected value of the selected program variable at a dependency point in the software program; and making the runtime value of the silent guard variable dependent on the runtime value of the selected program variable at the dependency point (pg. 18, section 7.1.3, especially 6th paragraph; pg. 19, fig. 18(a-e), A, B and C are translated into a1, a2, b1, b2, c1, c2, and x is the runtime value of the silent guard).

40. As per claim 69, the rejection of claim 68 under 35 USC 103(a) as being unpatentable over Collberg in view of Johnson is incorporated herein. (supra) Collberg further discloses the step of making the value of the silent guard variable dependent on the runtime value of the selected program variable comprises: computing the runtime value of the silent guard variable using a mathematical expression including the runtime value of the selected program variable and the expected value of the selected program variable at the dependency point (pg. 19, fig. 18(e), steps 5', 7' and 8').

41. As per claim 70, the rejection of claim 67 under 35 USC 103(a) as being unpatentable over Collberg in view of Johnson is incorporated herein. (supra) In addition, Collberg further discloses a method of converting static variables into procedural data as an obfuscation technique. In an example, a constant string within a selected computation is replaced by a function wherein a parameter of the function determines the string generated by the function (pgs. 18-19, sections 7.1.4 and fig. 19). Furthermore, this function is effectively a mathematical expression dependent on the variable parameter and evaluates to the constant string value if the run time value of the variable parameter is equal to the expected value of the variable parameter. Therefore, it would be obvious to one of ordinary skill in the art at the time the invention was made for the step of revising the selected computation to comprise the steps: selecting a constant value used in selecting computation; and replacing the constant value with a mathematical expression that is dependent on the runtime of the silent guard variable, such that the mathematical expression evaluates to the constant value if the runtime

value of the silent guard variable is equal to the expected value of the silent guard variable. One would be motivated to incorporate these steps since it prevents unscrupulous users from identifying the operation of the software (Collberg, ibid). The aforementioned cover the limitations of claim 70.

42. As per claim 71, the rejection of claim 67 under 35 USC 103(a) as being unpatentable over Collberg in view of Johnson is incorporated herein. (supra) Collberg further discloses the step of revising the selected computation comprises: selecting a computation variable used in the selected computation; determining an expected value of the computation variable at the execution point of the selected computation; and replacing the computation variable with a mathematical expression that is dependent on the runtime value of the silent guard variable, such that the mathematical expression evaluates to the expected value of the computation variable if the runtime value of the silent guard variable is equal to the expected value of the silent guard variable (pg. 19, fig. 18(e), steps 5', 7' and 8').

43. As per claim 72, the rejection of claim 67 under 35 USC 103(a) as being unpatentable over Collberg in view of Johnson is incorporated herein. (supra) Collberg further discloses the step of revising the selected computation comprises: inserting a mathematical expression including the runtime value of the silent guard variable and the expected value of the silent guard variable into the selected computation (pg. 19, fig. 18(e), steps 5'-10')

44. As per claim 79, the rejection of claim 78 under 35 USC 102(b) as being anticipated by Johnson is incorporated herein. (supra) Johnson does not disclose for the step of revising the selected computation to comprise the steps: selecting a constant value used in selecting computation; and replacing the constant value with a mathematical expression that is dependent on the runtime of the program variable, such that the mathematical expression evaluates to the constant value if the runtime value of the program variable is equal to the expected value of the program variable. Collberg discloses a method of converting static variables into procedural data as an obfuscation technique. In an example, a constant string within a selected computation is replaced by a function wherein a parameter of the function determines the string generated by the function (pgs. 18-19, sections 7.1.4 and fig. 19). Furthermore, this function is effectively a mathematical expression dependent on the variable parameter and evaluates to the constant string value if the run time value of the variable parameter is equal to the expected value of the variable parameter. Therefore, it would be obvious to one of ordinary skill in the art at the time the invention was made for the step of revising the selected computation to comprise the steps: selecting a constant value used in selecting computation; and replacing the constant value with a mathematical expression that is dependent on the runtime of the program variable, such that the mathematical expression evaluates to the constant value if the runtime value of the program variable is equal to the expected value of the program variable. One would be motivated to incorporate these steps since it prevents unscrupulous users from identifying the

operation of the software (Collberg, ibid). The aforementioned cover the limitations of claim 79.

45. As per claims 88 and 90, the rejection of claims 67-72 as being unpatentable over Collberg in view of Johnson is incorporated herein. (supra) In addition, Collberg discloses inserting dead or irrelevant code into a basic block, and also interleaving methods to introduce transformations to increase the complexity of the code and hence enhance the level of obfuscation. (pg. 11, section 6.2.1 “Insert Dead or Irrelevant Code”; pg. 15, section 6.3.2, “Interleave Methods”) Therefore, it would be obvious to one of ordinary skill in the art at the time the invention was made for the dependency point to be separated from the execution point by a plurality of program instructions. One would be motivated to do so to increase the level of obfuscation of the code as taught by Collberg, ibid. The aforementioned cover the limitations of claims 88 and 90.

Allowable Subject Matter

Claims 94, 95, 104 and 105 are allowed.

Claims 86 and 87 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

Conclusion

THIS ACTION IS MADE FINAL. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Communications Inquiry

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Jung W. Kim whose telephone number is 571-272-3804. The examiner can normally be reached on M-F 9:00-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Gilberto Barron can be reached on 571-272-3799. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Jk
March 5, 2007

GILBERTO BARRON JR
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100